Distributed Clustering for Ad Hoc Networks*

Stefano Basagni
Center for Advanced Telecommunications Systems and Services (CATSS)
Erik Jonsson School of Engineering and Computer Science EC33
The University of Texas at Dallas
2601 N. Floyd Rd.
Richardson, TX 75080
Tel. 972 883 2216 Fax. 972 883 2710
E-mail: basagni@utdallas.edu

Abstract

A *Distributed Clustering Algorithm* (DCA) and a Distributed Mobility-Adaptive Clustering (DMAC) algorithm are presented that partitions the nodes of a fully mobile network (*ad hoc* network) into *clusters*, thus giving the network a hierarchical organization. Nodes are grouped by following a new *weight*-based criterium that allows the choice of the nodes that coordinate the clustering process based on node mobility-related parameters. The DCA time complexity is proven to be bounded by a network parameter D_b that depends on the possibly changing *topology* of the network rather than on its *size*, i.e., the invariant number of the network nodes. The DMAC ...

Keywords: Clustering, *Ad Hoc* Networks, Wireless Mobile Networks, Distributed Network Algorithms.

1 Introduction

Obtaining a hierarchical organization of a network is a well-known and studied problem of distributed computing. It has been proven effective in the solution of several problems, such as, minimizing the amount of storage for communication information (e.g., routing and multicast tables), thus reducing information update overhead, optimizing the use of the network bandwidth, distributing resources throughout the network, etc.

In the case of *ad hoc networks*, i.e., wireless networks in which possibly all nodes can be mobile, partitioning the nodes into groups (*clusters*) is similarly important. In addition, *clustering* is crucial for controlling the spatial reuse of the shared channel (e.g., in terms of time division or frequency division schemes), for minimizing the amount of data to be exchanged in order to maintain routing and control information in a mobile environment, as well as for building and maintaining cluster-based *virtual* network architectures.

The notion of cluster organization has been used for ad hoc networks since their appearance. In [1, 3], a "fully distributed linked cluster architecture" is introduced mainly for hierarchical routing and to demonstrate the adaptability of the network to connectivity changes. With the advent of multimedia communications, the use of the cluster architecture for ad hoc network has been revisited by Gerla et. al. [4, 6, 5]. In these latter works the emphasis is toward the allocation of resources, namely, bandwidth and channel, to support multimedia traffic in an ad hoc environment.

*This work was supported in part by the Army Research Office under contract No. DAAG55-96-1-0382.

In existing solutions for *clustering* of ad hoc networks, the clustering is performed in two phases: clustering *set up* and clustering *maintenance*. The first phase is accomplished by choosing some nodes that act as coordinators of the clustering process (*clusterheads*). Then a *cluster* is formed by associating a clusterhead with some of its *neighbors* (i.e., nodes within the clusterhead's transmission range) that become the *ordinary nodes* of the cluster. Once the cluster is formed, the clusterhead can continue to be the local coordinator for the operations of its cluster (as in [1, 3, 5]), or, in order to avoid bottlenecks, the control can be distributed among the nodes of the cluster [4, 6]. The existing clustering algorithms differ on the criterium for the selection of the clusterheads. For example, in [1, 3, 6] the choice of the clusterheads is based on the unique identifier (ID) associated to each node: the node with the lowest ID is selected as clusterhead, then the cluster is formed by that node and all its neighbors. The same procedure is repeated among the remaining nodes, until each node is assigned to a cluster. When the choice is based on the maximum *degree* (i.e., the maximum number of neighbors) of the nodes, the algorithm described in [5] is obtained. A common assumption for the clustering set up is that the nodes *do not move* while the cluster formation is in progress. This is a major drawback, since in real ad hoc situations, no assumptions can be made on the mobility of the nodes.

Once the nodes are partitioned into clusters, the non mobility assumption is released, and techniques are described on how to maintain the cluster organization in the presence of mobility (clustering maintenance). For instance, in [1] a reorganization of the clusters, due to node mobility, is done periodically, just invoking the clustering process again (as at set up time). Of course, during the re-clustering process the network cannot rely on the cluster organization. Thus, this is a feasible solution only when the network does not need too many reorganizations. In [6] cluster maintenance in the presence of mobility is described where each node v decides locally whether to update its cluster or not. This decision is based on the knowledge of the v's one *and two* hop neighbors and on the knowledge of the local topology of v's neighbor with the *largest degree*. The resulting mobility-adaptive algorithm is thus different from the one used for the cluster formation (based on the nodes' IDs and on the knowledge of their one hop neighbors) and the obtained clustering has different properties with respect to the initial one.

In this paper we present a clustering algorithm suitable for both the clustering set up and maintenance that aims to overcome the above mentioned mobility related limitations. For our *Distributed Mobility-Adaptive Clustering* (DMAC, for short) algorithm we obtain the following properties, not available in previous solutions:

- Nodes can move, even during the clustering set up: DMAC is *adaptive* to the changes in the topology of the network, due to the mobility of the nodes or to node addition and/or removal.
- DMAC is fully *distributed*. A node decides its own role (i.e., clusterhead or ordinary node) solely knowing its current *one* hop neighbors.
- Every ordinary node always has *direct access* to at least one clusterhead. Thus, the nodes in the cluster are at most two hops apart. This guarantees fast intra-cluster communication and fast inter-cluster exchange of information between any pair of nodes.
- DMAC uses a *general* mechanism for the *selection of the clusterheads*. The choice is now based on generic *weights* associated with the nodes.
- The number of clusterheads that are allowed to be neighbors is a parameter of the algorithm (*degree of independence*).
- Finally, we define for DMAC a new weight-based criterium that allows the nodes to decide whether to change (switch) its role or not depending on the current condition of the network, thus minimizing the overhead due to the switching process.

Notice that the last three properties introduce the possibility for DMAC to be configured for the specific ad hoc network in which it operates. For example, the weight associated to a node allows us to express how that node is suitable for the role of clusterhead depending on properties of the node itself, such as its speed, its transmission power, etc. Moreover, previous solutions would not allow two clusterheads to be neighbors, forcing one of the two to resign. This is not always necessary, and two or more clusterheads should be allowed to be neighbors depending on specific network conditions and applications. The same reasoning applies when an ordinary node becomes a neighbor of another clusterhead: the criterium with which it chooses whether to affiliate with the new neighboring clusterhead or not should depend again on the current conditions of the network, and on the specific applications that use the cluster organization. We demonstrate the advantage of our new weight-based setting by presenting simulation results that compare DMAC with the "lowest ID first" algorithm of [6]. We show up to an 85% reduction on the communication overhead associated with cluster maintenance.

The paper is organized as follows. In the next section we define a model for ad hoc network and the clustering we want to obtain for these networks. The distributed clustering algorithm which adapts to changes in the network topology (DMAC) is then introduced and proved correct in Section 4. The paper concludes with some simulation results. The detailed codes of the procedures that implement DMAC can be found in Appendix A.

The methods used so far for obtaining physical clustering in ad hoc networks all implement some type of *greedy algorithm* for finding a set of nodes that act as coordinators of the clustering process (*clusterheads*). Once the clusterheads are selected, clusters are defined by associating each non-clusterhead node with a clusterhead following a specific rule. For instance, in the "maximum degree first" approach of Gerla et al. [5], a node with maximum *degree* (i.e., with the maximum number of neighbors) is selected as a clusterhead, then a cluster is formed by that node and all its neighbors. The same procedure is repeated among the nodes not assigned to a cluster yet, until all nodes belong to a cluster. In the "lowest ID first" method, used in Ephremides et al. [3] and in Gerla et al. [4, 6], node IDs are used to choose the clusterheads.

In this paper we present a *Distributed Clustering Algorithm* (DCA) that generalizes the previous approaches by allowing the choice of the clusterheads based on a generic *weight* (a real number ≥ 0) associated to each node: The bigger the weight of a node, the better that node for the role of clusterhead. The main advantage of this approach is that now, by representing with the weights node mobility-related parameters, we can choose for the role of clusterhead those nodes that are better suited for that role. For instance, when the weight of a node is inversely proportional to its speed, the less mobile nodes are selected to be clusterheads. Since these nodes either do not move or move slower than the other nodes, their cluster is guaranteed to have a longer life, and consequently the overhead associated with the cluster maintenance in the mobile environment is minimized. We prove the correctness of the DCA and we study its time and message complexities. In particular, we prove that the time complexity of the DCA is bounded by a network parameter that depends on the network *topology* (that may change due to nodes mobility) rather than on the *size* of the network, i.e., the invariant number of its nodes. By using simulations we demonstrate that the combination of the new weight-based clusterhead selection mechanism and the topology-dependent upper bound on the DCA time complexity yields a logarithmic bound on the DCA time complexity. This result improves exponentially the upper bound on the time complexity of distributed clustering presented in [6], which is claimed to be linear in the size of the network.

2 Preliminaries and problem definition

We model an *ad hoc* network by an undirected graph $G = (V, E)$ in which V, $|V| = n$, is the set of (wireless) nodes and there is an edge $\{u, v\} \in E$ if and only if u and v can mutually receive each others' transmission (this implies that all the links between the nodes are bidirectional). In this case we say that u and v are

neighbors. The set of the neighbors of a node $v \in V$ will be denoted by $\Gamma(v)$. Due to mobility, the graph can change in time.

Every node v in the network is assigned a unique identifier (ID). For simplicity, here we identify each node with its ID and we denote both with v. Finally, we consider weighted networks, i.e., a weight w_v (a real number ≥ 0) is assigned to each node $v \in V$ of the network. For the sake of simplicity, in this paper we stipulate that each node has a different weight. As an example, the topology of a simple ad hoc network is shown in Figure 1 (a).

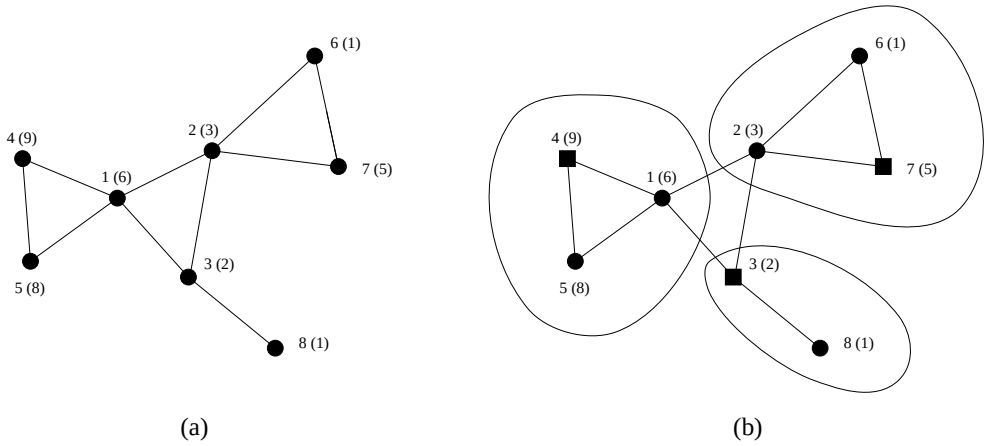

Figure 1 (a) An ad hoc network G with nodes v and their weights (w_v), $1 \leq v \leq 8$, and (b) a correct clustering for G.

Clustering an ad hoc network means partitioning its nodes into *clusters*, each one with a *clusterhead* and (possibly) some *ordinary nodes*. The choice of the clusterheads is here based on the *weight* associated to each node: the bigger the weight of a node, the better that node for the role of clusterhead. In order to meet the requirements imposed by the wireless, mobile nature of these networks, a clustering algorithm is required to partition the nodes of the network so that the following *ad hoc clustering properties* are satisfied:

1. Every ordinary node has at least a clusterhead as neighbor (*dominance* property).
2. Every ordinary node affiliates with the neighboring clusterhead that has the bigger weight.
3. No two clusterheads can be neighbors (*independence* property).

Property 1. is necessary to ensure that each ordinary node has direct access to at least one clusterhead (the one of the cluster to which it belongs), thus allowing fast intra- and inter-cluster communications. The second property ensures that each ordinary node always stays with the neighboring clusterhead with the bigger weight, i.e., with the clusterhead that can give it a "guaranteed good" service. Finally, property 3. guarantees that the network is covered by a "well scattered" set of clusterheads, so that each node in the network has a clusterhead in its neighborhood and it has direct access to that clusterhead.

Furthermore, given the possibly frequent changes in the network topology due to nodes' mobility, the algorithm is required to be executed at each node (i.e., the algorithm should be *distributed*) with the sole knowledge of the topology local to each node. Figure 1 (b) illustrates a correct clustering for the ad hoc network of Figure 1 (a) (the clusterheads of each cluster are the squared nodes).

3 A Distributed Clustering Algorithm (DCA)

In this section we describe a distributed algorithm that, given any ad hoc network, sets up a clustering that satisfies the properties listed in the previous section. The main assumption that we make here is that, during the execution of the algorithm, the network topology does not change (see also []). We also make two other common operational assumptions, namely, we assume that a message sent by a node is received correctly within a finite time (a *step*) by *all* its neighbors, and that every node knows its ID, its weight and the IDs and the weights of its neighbors.

The algorithm is executed at each node in such a way that a node v decides its own role (clusterhead or ordinary node) depending solely on the decision of its neighbors with bigger weights. Thus, initially, only those nodes with the bigger weight in their neighborhood will broadcast a message to their neighbors stating that they will be clusterheads. On receiving one or more of this "clusterhead" messages, a node v will decide to join the cluster of the neighboring clusterhead with the biggest weight. If no node with bigger weight have sent such a message (thus indicating that it is going to join some other cluster as an ordinary node), then v will send a clusterhead message. We will show that all the nodes terminate the algorithms being either clusterheads or ordinary nodes, and that the ad hoc clustering properties are satisfied.

Except for the initial routine, the algorithm is message driven: a specific procedure will be executed at a node depending on the reception of the corresponding message. We use two types of messages: CH(v), used by a node v to make its neighbors aware that it is going to be a clusterhead, and JOIN(v,u), with which a node v communicates to its neighbors that it will be part of the cluster whose clusterhead is node $u \in \Gamma(v)$. In the procedures, we use the following notation:

- v, the generic node executing the algorithm (from now on we will assume that v encodes not only the node's ID but also its weight w_v).
- *Cluster* (v), the set of nodes in v's cluster. It is initialized to $\emptyset$, and it is updated only if v is a clusterhead.
- *Clusterhead* , the variable in which every node records the (ID of the) clusterhead that it joins. It is initialized to **nil**.
- *Ch* $(-)$ and *Join* $(-,-)$, boolean variables. Node v sets *Ch* (u), $u \in \{v\} \cup \Gamma(v)$, to **true** when either it sends a CH(v) message ($v = u$) or it receives a CH(u) message from u ($u \neq v, u \in \Gamma(v)$). The boolean variable *Join* (u,t), $u \in \Gamma(v)$, $t \in V$, is set to **true** by v when it receives a JOIN(u,t) message from u. They are initialized to **false**.
- By executing the command EXIT, a node v quits the execution of the clustering algorithm (this means that it will not listen to any more messages from the network).
- With $\bigwedge$ and $\bigvee$ we indicate the conventional boolean operations "**and**" and "**or**," respectively.

Every node starts the execution of the algorithm at the same time, running the procedure *Init*. Only those nodes that have the biggest weight among all the nodes in their neighborhood will send a CH message ("init nodes"). Given the nature of the weights (real numbers), there always exists at least a node v that transmits the message CH(v). All the other nodes just wait to receive a message.

```
PROCEDURE Init;
begin
    if ⋀_{u∈Γ(v)} (w_v > w_u)
        then begin
                send CH(v);
```

```
                    Cluster (v) := Cluster (v) ∪ {v};
                    Ch (v) := true;
                    Clusterhead := v
                  end
end;
```

Then, we have the following two message triggered procedures:

• On receiving a CH message from a neighbor u, node v checks if it has received from *all* its neighbors z such that $w_z > w_u$, a JOIN(z,x) message, $x \in V$ (this is recorded in the corresponding boolean variables).[1] In this case, v will not receive a CH message from these z, and u is the node with the biggest weight in v's neighborhood that has sent a CH message. Thus, v joins u, and quits the algorithm execution (it already knows the cluster to which it belongs, i.e., its clusterhead). If there is still at least a node z, $w_z > w_u$, that has not sent a message yet, node v just records in the variable $Ch(u)$ that u sent a CH message, and keeps waiting for a message from z.

```
On receiving CH(u);
begin
    Ch (u) := true;
    if ⋀_{z∈Γ(v):w_z>w_u} (⋁_{x∈V} Join (z,x))
        then begin
                Clusterhead := u;
                send JOIN(v,Clusterhead);
                EXIT
             end
end;
```

• On receiving a JOIN(u,t) message, node v checks if it has previously sent a CH message (i.e., if it has already decided to be a clusterhead: when this happens, $Ch(v)$ is always assigned **true**). If this is the case, it checks if node u wants to join v's cluster ($v = t$), and possibly updates its $Cluster(v)$ set. Then, if all v's neighbors z such that $w_z < w_v$ have communicated their willingness to join a cluster, v quits the execution of the DCA. Notice that, in this case, node v does not care about its neighbors y (if any) such that $w_y > w_v$, because these nodes have surely joined a node $x \in V$ such that $w_x > w_v$ (thus permitting v to be a clusterhead). If node v has not sent a CH message, before deciding what its role is going to be, it needs to know what *all* the nodes z such that $w_z > w_v$ have decided for themselves. If v has received a message from all such nodes, then it checks the nature of the messages received. If they are all JOIN messages, this means that all those neighbors z have decided to join a cluster as ordinary nodes. This implies that now v is the node with the biggest weight among the nodes (if any) that have still to decide what to do. In this case, v will be a clusterhead, and it executes the needed operations (i.e., it sends a CH message, it updates its $Cluster(v)$ set, it sets its $Ch(v)$ to **true** and *Clusterhead* to v). At this point, v also checks if each neighbor y such that $w_y < w_v$ has already joined another cluster. If this is the case, v quits the algorithm execution: it will be the clusterhead of a cluster with a single node. Alternatively, if v has received at least a CH message from z, then it joins the cluster of the neighbor with the biggest weight that sent a CH message (this is selected by the means of the operator $\max_{w_z}\{z : Ch(z)\}$), and quits the execution of the DCA (notice that a node always quits the algorithm execution as soon as it sends a JOIN message).

```
On receiving JOIN(u,t);
begin
    Join (u,t) := true;
    if Ch (v)
```

[1] We stipulate that the boolean conditions $\bigwedge(\ldots)$ in the CH(u) and in the JOIN(u,t) procedures, evaluates to **true** when the sets on which they are based are the empty set.

```
then begin
        if v = t then Cluster(v) := Cluster(v) ∪ {u};
        if ⋀_{z∈Γ(v):w_z<w_v} (⋁_{x∈V} Join(z,x)) then EXIT
      end
else if ⋀_{z∈Γ(v):w_z>w_v} (Ch(z) ∨ (⋁_{x∈V} Join(z,x)))
        then if ⋀_{z∈Γ(v):w_z>w_v} (⋁_{x∈V} Join(z,x))
                then begin
                       send CH(v);
                       Cluster(v) := Cluster(v) ∪ {v};
                       Ch(v) := true;
                       Clusterhead := v;
                       if ⋀_{z∈Γ(v):w_z<w_v} (⋁_{x∈V} Join(z,x)) then EXIT
                     end
                else begin
                       Clusterhead := max_{w_z}{z : Ch(z)};
                       send JOIN(v,Clusterhead);
                       EXIT
                     end
end;
```

EXAMPLE 1. Let us consider the simple ad hoc network of Figure 1 (a). At time step 1 (beginning of the clustering process) all the nodes execute the *Init* procedure of the DCA. Nodes 4 and 7, being the nodes with the bigger weight in their neighborhood, send a CH message, declaring that they will be clusterheads. By the end of the same time step, nodes 1 and 5 receive the message CH(4), nodes 2 and 6 receive the message CH(7) and nodes 3 and 8 neither receive nor send any message. At time step 2, by executing the procedure On receiving CH(4), nodes 1 and 5 sends messages JOIN(1,4) and JOIN(5,4), respectively, and quit the execution of the DCA. (They are not prevented to do so by any node with weight bigger than the weight of node 4.) By the end of the same time step, node 4 has received the two JOIN messages from nodes 1 and 5 and, having received a JOIN message from any node with weight smaller than its own weight, quits the execution of the DCA: cluster $\{4,1,5\}$ is formed. By the end of time step 1 nodes 2 and 6, by executing the procedure On receiving CH(7), know that 7 is a neighboring clusterhead. Since there is no node in 6's neighborhood with weight $> w_7$, node 6 sends the message JOIN(6,7) and quits the execution of the DCA. Node 2 cannot do the same, since it has to wait for a message from node 1 (whose weight is $> w_7$). The JOIN(1,4) message is received at node 2 by the end of the second time step: since node 1 is going to join node 4's cluster, node 2 has to affiliate with node 7. Thus, by executing the procedure On receiving JOIN(1,4), node 2 sends the message JOIN(2,7). By the end of the third time step node 7 has received a message from all the nodes with weight smaller than its own weight and quits the execution of the DCA: cluster $\{7,2,6\}$ is formed. Let us consider now node 3. By the end of time step 2, it has received the message JOIN(1,4). At this time, though, node 3 cannot decide what role it is going to assume, since there is still another node (node 2) with a weight bigger than its own weight w_3 that has to decide. At the end of the third time step, node 3 finally receives the message JOIN(2,7) and, by executing the procedure On receiving JOIN(2,7), it realizes that all the nodes with weight bigger than its own weight have decided to join another cluster. Thus, it sends a CH(3) message, declaring that it will be a clusterhead. By the end of step 4 this message is received by node 8 (by executing the procedure On receiving CH(3)) that, since $w_3 > w_8$ and no other node with weight $> w_3$ is in its neighborhood, joins node 3's cluster by sending the message JOIN(8,3) and quits the execution of the DCA. As soon as node 3 receives the JOIN message from node 8 (by executing the procedure On receiving JOIN(8,3)) it also quits the execution of the DCA. Thus, by the end of the fifth step cluster $\{3,8\}$ is formed and the DCA is correctly terminated (see Figure 1 (b)). ◇

We prove now the time and message complexities of the DCA, and its correctness. We start by introducing some definitions.

Let us consider the *directed acyclic* graph $\vec{D} = (V, \vec{E})$ induced on $G = (V, E)$ by the following rule: $(v,u) \in \vec{E}$ if and only if $\{v,u\} \in E$ and $w_v < w_u$. Since there is an arc from v to u if and only if u's weight is bigger than the weight of v, it is clear that any *path* $v_{i_1} v_{i_2} \ldots v_{i_k}$ in $\vec{D}$, $k \leq n-1$, is such that

$w_{v_{i_1}} < w_{v_{i_2}} < \ldots < w_{v_{i_k}}$. With $v \to u$ we indicate the *longest path* in $\vec{D}$ from node v to node u (if any), and with $|v \to u|$ we denote its length, i.e., the number of the arcs in the path. (If there is no path in $\vec{D}$ from v to u we define $|v \to u| = 0$.) Let us now consider the set $I = \{i : i \in V, w_i > w_u, u \in \Gamma(i)\}$ of the init nodes (they are all and only the nodes for which no arc (i,v) exists in $\vec{E}$, $v \in V \setminus I$). Of course, $I \neq \emptyset$. For each node $v \in V \setminus I$, let us define the following *blocking distance* function $b : V \setminus I \to \{1, \ldots, n-1\}$ such that:

$$v \mapsto b(v) = \max\{|v \to i| : i \in I\}.$$

The function b is extended to the init nodes by defining $b(i) = \max\{b(v) : v \in \Gamma(i)\}$. Notice that b is well defined because at least one path from $v \notin I$ to at least one init node always exists. Intuitively, the blocking function $b(v)$ indicates how many time steps node v is blocked in deciding which role it is going to have. Finally, with $D_b = \max\{b(v) : v \in V\}$, we indicate the *blocking diameter* of the network.[2] Of course, $D_b \leq n-1$. The blocking diameter of the network of Figure 1 is 5 (which is $b(8)$, the length of the longest path from node 8 to the init node 4 formed by nodes 8, 3, 2, 1, 5 and 4).

The following result is fundamental in proving the time and message complexities of the DCA.

Proposition 1 *Each node of the network sends exactly one message within $D_b + 1$ steps.*

Proof *Each node v sends at least a message within $D_b + 1$ steps.* We proceed by induction on $b(v) = \ell \leq D_b$, $v \in V$, showing that if $b(v) = \ell$, then v sends either a CH or a JOIN message within $\ell + 1$ steps. Let $b(v) = 1$. If $v \in I$ then the thesis trivially holds at the first step. If $v \notin I$, then v receives all the CH(i) message(s), $i \in I$, in the first step. In this case, we observe that all the neighbors of v that are not init nodes have weights $< w_v$ (otherwise, a node $u \in \Gamma(v)$ such that $w_u > w_v$ and $u \notin I$ has to have at least a neighbor z such that $w_z > w_u$, and this implies $b(v) > 1$). Thus, executing the CH procedure, v sends a JOIN(v,i) message to the init node with the biggest weight (and terminates) in the second step.

Let us assume now that each node with blocking distance ℓ, $1 < \ell < D_b$, sends a message within $\ell + 1$ steps. Let v be a node such that $b(v) = \ell + 1$. By definition of b, all the nodes $u \in \Gamma(v)$ with $w_u > w_v$ are such that $b(u) \leq \ell$. Thus, by inductive hypothesis, they all send a message within $\ell + 1$ steps. If all these messages are JOIN messages then, within $\ell + 2$ steps v sends a CH message, otherwise it sends a a JOIN message. Thus, by induction, every node v such that $b(v) = \ell$ sends a message CH or JOIN within $\ell + 1$ steps, $1 \leq \ell \leq D_b$.

Each node v sends at most one message. Suppose v has sent a CH(v) message. Then:

- v cannot send another CH(v) message. The only way in which v can send a CH message are either while executing the *Init* procedure or while executing the **then** branch on the innermost **if** of the JOIN procedure. The first case is impossible because the *Init* procedure can be executed just once. The second case is also prevented to happen because the $Ch(v)$ variable is set to **true** when the first CH message has been sent.

- v cannot send a JOIN(v,u) message. Let us suppose that v sends a JOIN(v,u) message. The only case in which this is possible is by executing the CH(u) procedure, $u \in \Gamma(v)$ (as above, the $Ch(v)$ variable has been assigned **true** when the first CH message has been sent, and this prevents the execution of the **else** branch of the outermost **if**). We notice that a node $u \in \Gamma(v)$ such that $w_u < w_v$ can never send a CH(u) message, neither using the *Init* procedure (v prevents it to issue such a message) nor executing the **then** branch on the innermost **if** of the JOIN procedure (node u has already received the CH(v) message, so that it is prevented to execute that branch). Thus, we consider a node u such that $w_u > w_v$. In this case, we notice that v did not send the CH(v) message by executing the *Init* procedure

[2] The blocking diameter D_b is not to be confused with the network diameter D, which is defined as the length of the longest among the *shortest* paths. In general, there is no relation between D_b and D.

(u prevents it to issue such a message). Therefore, v has sent that message by executing the **then** branch on the innermost **if** of the JOIN procedure. This is possible only if v has received a JOIN(u,x) message, $x \in V$, *before* receiving the CH(u) message. But this is impossible because, in this case, u would have quit the DCA (so that it cannot send the CH(u) message).

When v sends a JOIN message, it always quits the execution of the DCA. So it is impossible for it to send another message. •

From this result we can state the time complexity of the DCA.

Corollary 1 *The DCA terminates within $D_b + 1$ steps.*

Proof Proposition 1 ensures that within $D_b + 1$ steps every ordinary node of the ad hoc network terminates the algorithm. This guarantees that within the same number of steps every clusterhead also terminates the DCA (see procedure On receiving JOIN(u,t)). •

On the ad hoc network of Figure 1 (a) the DCA terminates (correctly) in $5 < D_b + 1 = 6$ time steps (see Example 1). We notice that, having bound the time complexity of the DCA to a parameter D_b that depends on the (possibly changing) topology of the ad hoc network as opposed to an invariant like the number n of its nodes, induces an upper bound on the DCA time complexity that improves on the $O(n)$ upper bound presented in [6] in all those cases in which $D_b \ll n$.[3],

REMARK 1. If we consider algorithms that produce a clustering with the three properties stated in Section 2 and for which at least a message has to be sent by each node of the network, the upper bound induced by the previous corollary cannot be improved. Indeed, it is easy to find networks in which the DCA needs exactly $D_b + 1 = n$ steps to terminate (e.g., consider a network that is a blocking path with n nodes, n odd). ⋆

The following corollary states the message complexity of the DCA.

Corollary 2 *The message complexity of the DCA is n.*

Corollary 3 *Each node belongs exactly to a cluster.*

Proof Each node v that sends a CH(v) message belongs to the cluster of which it is the clusterhead. On the other hand, if a node u sends a JOIN(u,t) message, $t \in \Gamma(u)$, then it belongs to the cluster whose clusterhead is t. •

The previous corollary immediately implies that a node is either a clusterhead or an ordinary node, that an ordinary node joins only one clusterhead (no overlapping clusters), that an ordinary node is only one hop apart from the clusterhead it joins (dominance property, see 1., page 4), and thus that the diameter of a cluster is at most two. These properties are all independent on the weighting associated to the network. The following proposition takes weights into account, characterizing uniquely the DCA.

Proposition 2 *A node v of an ad hoc network sends a* CH *message if and only if all its neighbors z with $w_z > w_v$ have already joined another clusterhead. A node v sends a* JOIN(v,u) *message if and only if it has at least a neighbor u with $w_u > w_v$ that has sent a* CH(u) *message, all the neighbors z such that $w_z > w_u$ have joined a cluster, and the other neighbors y ($w_v < w_y < w_u$) have sent either a* CH *or a* JOIN *message. In this case, u is guaranteed to have the biggest weight among all nodes that have sent a* CH *message.*

[3] We notice that, even in the case $D_b = n - 1$, the time complexity of the proposed algorithm is *exactly* $D_b + 1$ steps, i.e., our result in not asymptotic.

Proof Let us consider the set $Z = \{z : z \in \Gamma(v), w_z > w_v\}$. If $Z = \emptyset$ then the thesis follows because of the Init procedure (node v sends a CH(v) message). When $Z \neq \emptyset$, then the only case in which v can send a CH(v) message is by executing the **then** branch of the innermost **if** in the JOIN procedure. This can happen if and only if $\bigwedge_{z \in Z}(\bigvee_{x \in V} Join(z,x)) =$ **true**, i.e., if and only if all the nodes $z \in \Gamma(v)$, $w_z > w_v$, have already joined a clusterhead $x \in V$.

A node v can send a JOIN message if and only if it executes either the CH procedure or the **else** branch of the innermost **if** of the JOIN procedure. In both cases it has to have received a CH message from at least a node u such that $w_u > w_v$, and JOIN messages from all the nodes with weights $> w_u$. In the case of the JOIN procedure, v has also to receive (either CH or JOIN) messages from all those nodes y such that $w_v < w_y < wu$. If the JOIN message has been sent while executing the CH(u) procedure, u is guaranteed to have the biggest weight among all the nodes $z \in \Gamma(v), w_z > w_v$, because all nodes with weight $> w_u$ have already sent a JOIN message (otherwise, v could not execute the **then** branch of the **if** command). If the JOIN message has been sent while executing the **else** branch of the innermost **if** in the JOIN procedure, the thesis is guaranteed by the max operator. •

Corollary 4 *No two clusterheads can be neighbors.*

Proof Let v and u be neighboring clusterheads. Suppose, without loss of generality, that $w_u > w_v$. Then, v cannot be a clusterhead, because there exists at least one of its neighbor, namely u, with $w_u > w_v$, that has not joined another clusterhead (u has already sent a CH(u) message, and it cannot send a JOIN message anymore). •

Proposition 2 and the previous corollary state the clustering properties 2. and 3., respectively (Section 2). The previous results are summed up in the following theorem that states the correctness of the DCA.

Theorem 1 *All the nodes of the network exit the execution of the algorithm having been assigned either the role of clusterhead or the role of ordinary node so that the three ad hoc clustering properties are satisfied.*

We conclude this section by noticing that the requirement that all the nodes initiate the DCA at the same time is not a severe constraint. In order to "synchronize" the nodes on a specific initial time it is enough that a selected node issues a broadcast message to request the start of the clustering process. Broadcast algorithms such as the one presented in [2] are suitable algorithms for this purpose, being completely distributed, deterministic and mobility adaptive. In this way, it is always guaranteed that after a specific bounded amount of time each node of the network knows that a clustering has been requested, and the DCA can start at each node.

4 Distributed and Mobility-Adaptive Clustering (DMAC)

In this section we describe a distributed algorithm for the set up and the maintenance of a cluster organization in the presence of nodes' mobility that satisfies the three ad hoc clustering properties. The main difference with the DCA presented in the previous section consists in the fact that here we do not assume that during the clustering process the nodes of the network need need not to move. This makes this algorithm suitable for both the clustering set up and its maintenance, which was not available in previous solutions. Adaptation to changes in the network topology is now made possible by letting each node to properly "react" not only to the reception of a message from other nodes, but also to the the failure of a link with another node (possibly caused by a node failure, or by nodes' movements) or to the presence of a new link.

In describing the procedures of our Distributed and Mobility-Adaptive Clustering (DMAC) algorithm, we still assume that a message sent by a node is received correctly within a finite time (a step) by all its neighbors. We also assume that each node knows its own ID, its weight, its role (if it has already decided to be a clusterhead or an ordinary node) and the ID, the weight and the role of all its neighbors (if they have already decided their role). When a node has not yet decided what its role is going to be, it is considered as an ordinary node.

Except for the procedure that each node executes as soon as it starts the clustering operations, as in the case of DCA, the algorithm is message driven. Here, we use the same two types of messages used in the DCA (namely, CH(v) and JOIN(v,u)).

In the following we use the notation for v, *Cluster* (v), *Clusterhead* and *Ch* $(-)$ with the same meaning that they have in Section 3. Furthermore, we assume that:

- Every node is made aware of the failure of a link, or of the presence of a new link by a service of a lower level (this will trigger the execution of the corresponding procedure);
- The procedures of DMAC (M-procedures, for short) are "atomic," i.e., they are not interruptible;
- At clustering set up or when a node is added to the network its variables *Clusterhead* , *Ch* $(-)$, and *Cluster* $(-)$ are initialized to **nil**, **false** and $\emptyset$, respectively.

The following is the description of the six M-procedures.

• *Init*. At the clustering set up, or when a node v is added to the network, it executes the procedure *Init* in order to determine its own role. If among its neighbors there is at least a clusterhead with bigger weight, then v will join it. Otherwise it will be a clusterhead. Notice that a neighbor with a bigger weight that has not decided its role yet (this may happen at the clustering set up, or when two or more nodes are added to the network at the same time), will eventually send a message (every node executes the *Init* procedure). If this message is a CH message, then v will affiliate with the new clusterhead.

```
PROCEDURE Init;
begin
   if {z ∈ Γ(v) : w_z > w_v ∧ Ch(z)} ≠ ∅
      then begin
              x := max_{w_z > w_v} {z : Ch(z)};
              send JOIN(v,x);
              Clusterhead := x
           end
      else begin
              send CH(v);
              Ch(v) := true;
              Clusterhead := v;
              Cluster(v) := {v}
           end
end;
```

• *Link_failure*. Whenever made aware of the failure of the link with a node u, node v checks if its own role is clusterhead and if u used to belong to its cluster. If this is the case, v removes u from *Cluster* (v). If v is an ordinary node, and u was its own clusterhead, then it is necessary to determine a new role for v. To this aim, v checks if there exists at least a clusterhead $z \in \Gamma(v)$ such that $w_z > w_v$. If this is the case, then v joins the clusterhead with the bigger weight, otherwise it becomes a clusterhead.

```
PROCEDURE Link_failure(u);
begin
    if Ch(v) and (u ∈ Cluster(v))
        then Cluster(v) := Cluster(v) \ {u}
        else if Clusterhead = u then
                if {z ∈ Γ(v) : w_z > w_v ∧ Ch(z)} ≠ ∅
                    then begin
                            x := max_{w_z > w_v} {z : Ch(z)};
                            send JOIN(v,x);
                            Clusterhead := x
                         end
                    else begin
                            send CH(v);
                            Ch(v) := true;
                            Clusterhead := v;
                            Cluster(v) := {v}
                         end
end;
```

• *New_link*. When node v is made aware of the presence of a new neighbor u, it checks if u is a clusterhead. If this is the case, and if w_u is bigger than the weight of v's current clusterhead, than, independently of its own role, v affiliates with u.

```
PROCEDURE New_link(u);
begin
    if Ch(u) then
        if (w_u > w_Clusterhead)
            then begin
                    send JOIN(v,u);
                    Clusterhead := u;
                    if Ch(v) then Ch(v) := false
                 end
end;
```

• *On receiving* CH(u). When a neighbor u becomes a clusterhead, on receiving the corresponding CH message, node v checks if it has to affiliate with u, i.e., it checks whether w_u is bigger than the weight of v's clusterhead or not. In this case, independently of its current role, v joins u's cluster.

```
On receiving CH(u);
begin
    if (w_u > w_Clusterhead) then begin
                                      send JOIN(v,u);
                                      Clusterhead := u;
                                      if Ch(v) then Ch(v) := false
                                  end
end;
```

• *On receiving* JOIN(u,z). On receiving the message JOIN(u,z), the behavior of node v depends on whether it is a clusterhead or not. In the affirmative, v has to check if either u is joining its cluster ($z = v$: in this case, u is added to $Cluster(v)$) or if u belonged to its cluster and is now joining another cluster ($z \neq v$: in this case, u is removed from $Cluster(v)$). If v is not a clusterhead, it has to check if u was its clusterhead. Only if this is the case, v has to decide its role: It will join the biggest clusterhead x in its neighborhood such that $w_x > w_v$ if such a node exists. Otherwise, it will be a clusterhead.

```
On receiving JOIN(u, z);
begin
  if Ch(v)
    then if z = v then Cluster(v) := Cluster(v) ∪ {u}
                  else if u ∈ Cluster(v) then Cluster(v) := Cluster(v) \ {u}
    else if Clusterhead = u then
           if {z ∈ Γ(v) : w_z > w_v ∧ Ch(z)} ≠ ∅
             then begin
                    x := max_{w_z > w_v} {z : Ch(z)};
                    send JOIN(v,x);
                    Clusterhead := x
                  end
             else begin
                    send CH(v);
                    Ch(v) := true;
                    Clusterhead := v;
                    Cluster(v) := {v}
                  end
end;
```

We conclude this section by showing that by using the M-procedures we obtain and maintain for any ad hoc network a clustering that always satisfies the ad hoc clustering properties listed in Section 2.

Theorem 2 *Using the M-procedures, any ad hoc network is (maintained) clustered in such a way that the ad hoc clustering properties are always satisfied.*

Proof We start by noticing that it is easy to check from the code of the *Init* procedure that as soon as a node has executed this procedure, it is always assigned a role which is consistent with the clustering properties.

We proceed by showing that by executing the M-procedures in reaction to changes in the network topology, the nodes assume/change their roles so that the ad hoc clustering properties are always satisfied.

1. That each ordinary node v does not affiliate with more that one clusterhead is evident by noticing that anytime it sends a JOIN(v,u) message its variable *Clusterhead* is initialized (only) to u. That v affiliates with at least a clusterhead derives from the fact that when it has to decide its role and there are clusterheads with bigger weights among its neighbors, or when a switch to another cluster is required, the ordinary node v always looks for the clusterhead with the biggest weight and affiliates with it. This can be easily checked in the codes of: the *Init* procedure (**then** branch); the *Link_Failure* and the On receiving JOIN(u,z) procedures (when the link with its clusterhead u is broken, or the clusterhead u has resigned, joining another node, v looks for another clusterhead; of course, if no clusterhead is available, it will be a clusterhead), and the *New_Link* and the On receiving CH(u) procedures (if u is the new clusterhead on the block, if node v needs to affiliate with u, it does so by executing the **then** branch of the innermost **if**). Thus, there is no case in which an ordinary node v remains without a clusterhead.

2. At the clustering set up, or when a node is added to the (already clustered) network, or when its current clusterhead moves away, a node always affiliates with the clusterhead with the biggest weight (if there is no such clusterhead, it will become a clusterhead itself), so that the second ad hoc clustering property is always satisfied (see the code of procedures *Init*, *Link_Failure*, On receiving JOIN). The other cases to consider are when an ordinary node v switches from a cluster to another, or when v is a clusterhead that resigns to join the cluster of a new neighboring clusterhead. In these cases, the second

property is guaranteed by the procedures *New_Link* and On Receiving CH, where node *v* switches to *u*'s cluster only if $w_u > w_{Clusterhead}$.

3. Each time a node becomes a clusterhead, i.e., it transmits a CH message (see the **else** branch of the if in the *Init* procedure, the **else** branch of the innermost if in the *Link_Failure* procedure, and the same branch in the On receiving JOIN procedure), it does so because there is no other neighboring clusterhead with which it can affiliate. The other cases that remain to be checked are when either a clusterhead *v* has one of its neighbors that becomes a clusterhead, or a clusterhead moves into its neighborhood, and the weight of the new neighbor does not force *v* to affiliate with it. In both these cases, the third ad hoc clustering property is guaranteed by the execution of the **else** branch

Thus, the three properties for ad hoc clustering are always satisfied. •

5 Conclusions

This paper presented two distributed algorithms, DCA and DMAC, for the efficient partitioning of the the nodes of an ad hoc wireless network into clusters with a clusterhead and some ordinary nodes. This is a practically important task, especially for all those network algorithms/applications that assume a mobility-adaptive hierarchical organization of the network. A new weight-based criterium is introduced for the cluster formation that allows the choice of the clusterheads based on node mobility-related parameters, not available in previous clustering algorithms. The proposed algorithms needs only knowledge of the local topology at each node (one hop neighbors), and allows each ordinary node to have direct access to at least a clusterhead, thus guaranteeing fast inter- and intra-cluster communication between each pair of nodes. The DCA is easy to implement and its time complexity is proven to be bounded by a network parameter that depends on the possibly changing topology of the ad hoc network rather than on *n*, the invariant size of the network. The DMAC combines easiness of implementation with full adaptation to the mobility of the nodes, even during clustering set up.

References

[1] BAKER, D. J., EPHREMIDES, A., AND FLYNN, J. A. The design and simulation of a mobile radio network with distributed control. *IEEE Journal on Selected Areas in Communications SAC-2*, 1 (January 1984), 226–237.

[2] BASAGNI, S., AND CHLAMTAC, I. Broadcast in peer-to-peer networks. In *Proceedings of the Second IASTED International Conference European Parallel and Distributed Systems, Euro-PDS'98* (Vienna, Austria, July 3–5 1998), O. Bukhres and H. El-Rewini, Eds., pp. 117–122.

[3] EPHREMIDES, A., WIESELTHIER, J. E., AND BAKER, D. J. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proceedings of the IEEE 75*, 1 (January 1987), 56–73.

[4] GERLA, M., AND LIN, C. R. Multimedia transport in multihop dynamic packet radio networks. In *Proceedings of International Conference on Network Protocols* (Tokyo, Japan, 7–10 November 1995), pp. 209–216.

[5] GERLA, M., AND TSAI, J. T.-C. Multicluster, mobile, multimedia radio network. *Wireless Networks 1*, 3 (1995), 255–265.

[6] LIN, C. R., AND GERLA, M. Adaptive clustering for mobile wireless networks. *Journal on Selected Areas in Communications 15*, 7 (September 1997), 1265–1275.